A Two-step Optimization Technique for Functions Placement, Partitioning, and Priority Assignment in Distributed Systems

Asma Mehiaoui, Ernest Wozniak
CEA LIST DILS
asma.mehiaoui, ernest.wozniak}@cea.fr

Sara Tucci-Piergiovanni, Chokri Mraidha
CEA LIST DILS
{sara.tucci, chokri.mraidha}@cea.fr

Haibo Zeng
McGill University
haibo.zeng@mcgill.ca

Marco Di Natale
Scuola Superiore Sant’Anna
marco@sssup.it

Jean-Philippe Babau
Lab-STICC, University of Brest
{jean-philippe.babau, laurent.lemarchand}@univ-brest.fr

Abstract

Modern development methodologies from the industry and the academia for complex real-time systems define a stage in which application functions are deployed onto an execution platform. The deployment consists of the placement of functions on a distributed network of nodes, the partitioning of functions in tasks and the scheduling of tasks and messages. None of the existing optimization techniques deal with the three stages of the deployment problem at the same time. In this paper, we present a staged approach towards the efficient deployment of real-time functions based on genetic algorithms and mixed integer linear programming techniques. Application to case studies shows the applicability of the method to industry-size systems and the quality of the obtained solutions when compared to the true optimum for small size examples.

Categories and Subject Descriptors C.3 [Real-time and embedded systems]; G.1.6 [Optimization]: linear programming; stochastic programming

Keywords Distributed real-time applications, response-time analysis, optimization, linear programming, genetic algorithm, placement, partitioning, scheduling

1. Introduction

In the development of real-time cyber-physical systems, abstraction levels are used to manage complexity [17]. Industrial standards (like the automotive AUTOSAR [2] and the Model-Driven Architecture from the OMG [1]) and academic frameworks (including the Platform-Based Design [27]) recommend system development along the lines of the Y-chart approach [15]: a functional model representing the system functions and the signals exchanged among them is deployed onto an execution platform model consisting of nodes, buses, tasks and messages. End-to-end real-time constraints (deadlines) are specified on transactions, that is, chains of functions, activated by an external event (e.g. as detected by a sensor) or a timer, and terminating with the execution of a sink function (e.g. sending a command to actuators) [2]. Classically as in AUTOSAR, two subsequent and separated activities (often carried out by different teams) are dedicated to functional deployment. First, the placement of functions on execution nodes is defined. Next, the partitioning of the set of functions and signals in tasks and messages and the scheduling of tasks and messages is computed. In fixed-priority scheduled systems, this includes the assignment of priorities. Once the functional level is arranged in tasks and messages, the worst-case timing behavior can be computed and compared against end-to-end deadlines. Ideally, the placement, partitioning, and scheduling problem should be automated by solving an optimization problem with respect to the metrics and real-time constraints, but because of its inherent complexity – placement, partitioning, and scheduling are NP-hard problems – suboptimal staged approaches and heuristics are used. The first stage (as in [17]) may be dedicated to placement in isolation, i.e. functions and signals are assigned to nodes and buses without the definition of the task and message model (including their priorities). As the deployment is only partial, worst-case latencies cannot be evaluated and the placement is evaluated by simple metrics and constraints, such as resource utilization. Other approaches lie in solving the placement and scheduling of tasks and messages with respect to latency-based constraints and metrics [8, 32]. In this case, the function-to-task and signal-to-message partitioning is previously solved in isolation with simple heuristics, such as grouping in one task all functions belonging to the same transaction and/or executing at the same rate or a one-to-one mapping between functions and signals to tasks and messages. Both approaches may lead to sub-optimal deployments.

In this paper we are interested in tackling function placement, partitioning and scheduling exploring the use of Mixed Integer Linear Programming (MILP) and Genetic Algorithms (GA). MILP and GA algorithms are complementary and may derive mutual benefits from a joint application to the problem. An MILP formulation is easily extensible, re-targetable to a different optimization metric and can easily accommodate additional constraints or legacy components. An MILP formulation may benefit from the application of powerful commercial solvers. The method also guarantees optimality in case the solver terminates (when the problem size is manageable) and the distance of the current solution from the optimum can be bounded at any time (computed as the gap between the current best solution and the one of the relaxed linear problem), thus it is possible to evaluate the quality of the intermediate solutions generated by the solver. GA solutions typically scale much better. However, the quality of the solution provided by GA depends on many factors, for example, the appropriate choice of a
crossover operator, and is very hard to evaluate. Using both techniques, the MILP solver can provide the true optimum for average size problems, helping in the tuning of the GA formulation and the assessment of its quality. For larger size problems, it can provide an upper bound to the optimum metric, helping in the evaluation of the quality obtained by the GA.

**Our Contributions.** In this paper we first present a MILP technique that solves the placement, partitioning and scheduling problem (from now on PPS) at the same time (unlike existing approaches). The MILP formulation of the problem can be solved and returns the optimal solution, but it is practically applicable only to small-size systems. In order to scale to industry-size systems we then propose a staged, divide-and-conquer approach with an iterative improvement optimization. Through divide-and-conquer the PPS is divided in two sub-problems of manageable size solved in cascade. Response-time optimization and latency constraints are considered for both sub-problems. This staged approach has been implemented using MILP and GA techniques. Of course, when a staged MILP’s solution is used, several benefits are lost, including the guarantee of optimality. However, the staged MILP solution provides a good tradeoff between scalability (a primary concern for industrial size systems) and the quality of solutions. Furthermore the MILP solution is useful to well-design the staged GA solution, which scales to even larger systems.

The paper is organized as follows. Section 2 presents the related work. Section 3 presents basic definitions and assumptions, then formally introduces the PPS problem. Section 4 presents the staged optimization strategy for the PPS problem and the two proposed formulations (MILP and GA) for the inner optimization stages. Section 5 shows the experimental results and Section 6 concludes the paper.

## 2. Related work

Most automotive controls are designed based on *run-time static priority scheduling* of tasks and messages. Examples of standards supporting this scheduling model are the AUTOSAR operating system [2], and the CAN bus [10] arbitration model.

Optimization techniques have been extensively used to find good solutions to deployment problems. [3] classifies 188 papers along many axes, i.e. design goals and constraints, degrees of freedom, problem solved. None of the surveyed papers, however, considers worst-case latency of the deployed transactions as an additional constraint or goal.

End-to-end deadlines have been discussed in research work in single-processor and distributed architectures. In transaction-based activation models (such as the holistic and jitter propagation model in [24, 28] and the transaction model with offsets in [21]), messages are queued by sender tasks and the arrival of messages at the destination node triggers the activation of the receiver task. In such models, task and message schedulers have cross dependencies because of the propagation of the activation signals and real-time analysis can be performed using the holistic model [28] [23] based on the propagation of the release jitter along the computation path. When offsets can be enforced for tasks and messages to synchronize activations, the system can be analyzed as in [21].

Despite advances in timing analysis, the optimized deployment synthesis problem has not received comparable attention. In [24] the authors discuss the use of genetic algorithms for optimizing priority and period assignments to tasks with respect to an extensibility metric and a number of constraints, including end-to-end deadlines and jitter. In [9], the authors describe a procedure for period assignment on priority-scheduled single-processor systems. In [22, 23], a heuristics-based design optimization algorithm for mixed time-triggered and event-triggered systems is proposed. The algorithm, however, assumes that nodes are synchronized. An integrated optimization framework is also proposed in [14] for systems with periodic tasks on a network of processor nodes connected by a time-triggered bus. The framework uses Simulated Annealing (SA) combined with geometric programming to hierarchically explore task allocation, task priority assignment, task period assignment and bus access configuration.

In [8], task allocation and priority assignment were defined with the purpose of optimizing the extensibility with respect to changes in task computation times. The proposed solution was based on simulated annealing. In [13], a generalized definition of extensibility on multiple dimensions (including changes in the execution times of tasks but also period speed-ups and possibly other metrics) was presented. Also, a randomized optimization procedure based on a genetic algorithm was proposed to solve the optimization problem. The focus is on the multi-parameter Pareto optimization, and how to discriminate the set of optimal solutions. Other works assume a communication-by-sampling model, in which tasks and messages are activated periodically and exchange information over shared variables. In this context, [32] provides an MILP formulation for the problem that considers the placement, scheduling and signal partitioning as degrees of freedom. The formulation is extended to consider extensibility in [30]. Azketa et al. used genetic algorithms to assign priorities to tasks and messages, then to map tasks and messages on the execution platform for event-triggered systems [6]. Finally, a mixed model in which information can be exchanged synchronously (with tasks and messages activating the successors, as in the transaction model) or by periodic sampling is considered in [29], where only the optimization of the activation modes is provided (task and message placement and priorities are fixed). A common characteristic of the above cited works is that function placement and partitioning is not considered.

Functions partitioning problem was considered in [7] [26] and [16] but only for a single-processor system, i.e., without the placement problem.

## 3. System model and the PPS problem

This section introduces the basic definitions and assumptions on the system model. Then, the PPS problem is defined and most significant parts of the MILP formulation are given.

### 3.1 System model

The considered system consists of a *physical architecture* (Network topology) and a *logical architecture* (Functional graph).

The network topology is represented by a graph $G_\Psi$ of execution nodes $C = \{c_1, c_2, \ldots, c_n\}$ connected through buses $B = \{b_1, b_2, \ldots, b_s\}$. Each node runs a real-time operating system with a preemptive fixed-priority scheduling (such as an AUTOSAR OS). Communication buses are assumed to be Controller Area Networks (CAN), arbitrated by priority (the identifier field of the message). The execution nodes and communication buses may have different processing and transmission speeds. Both execution nodes and communication buses have a maximal capacity utilization, respectively $\mu_a$ and $\mu_b$, that must not be exceeded. Each execution node offers a set of tasks that perform the computation required by the system functions. We denote the set of tasks offered by the $i$-th execution node as $T_i = \{t_1, t_2, \ldots, t_m\}$. Each task $t_i$ is characterized by a set of functions that it executes and a priority level $\pi_{t_i}$. We denote as $\tau_{t_i}$ the task to which $t_i$ belongs. In turn, communication buses offer a set of messages $\psi = \{m_1, m_2, \ldots, m_n\}$ that realize the transmission of signals between remote tasks. Each message $m_i$ is defined by the set of signals that it transmits and a priority level $\pi_{m_i}$.

The functional graph depicts the operational process of the system, in which events generated by sensors or by users trig-
gender the computation of a set of control functions or algorithms, which eventually define a system response. This behavior may be captured by a dataflow model, and described as a directed acyclic graph \( \zeta \) composed by a set of concurrent linear transactions \( \zeta_i = (\Gamma_i, \Pi_{\Gamma_i}, \Gamma_i) \). Each transaction is a 2-tuple \( \{ E, \rho \} \), where \( E = \{ f_1, f_2, ..., f_l \} \) is the set of functions that represent the atomic operations and \( \rho = \{ l_1, l_2, ..., l_f \} \) is a set of links representing the interactions between functions. Functions exchange data through signals on these links. \( \Phi = \{ s_1, s_2, ..., s_s \} \) denotes the set of signals. A transaction \( \Gamma_i \) is triggered by an external event \( e_i \) that can be periodic or sporadic with, respectively, an activation period or a minimum inter-arrival time, denoted in both cases as \( P_i \). Functions and signals within a transaction inherit their period (respectively \( P_j \) and \( s_n \)) from the activation period of the external event triggering this transaction. In addition, each transaction \( \Gamma_i \) has a deadline \( D_i \), which eventually define a system response. This behavior may consist in mapping each function into exactly one task and each signal into at most one message. \( A_{i,k} \) is a boolean variable set to 1 if function \( f_i \) (signal \( s_j \)) is placed on task \( t_k \) (message \( m_k \)). \( A_{i,j,k} \) is also a boolean, and denotes whether both \( f_i \) and \( f_j \) are partitioned to \( t_k \). The following constraints guarantee that each function is assigned to exactly one task and ensure the consistent definition of \( X_{i,j,k} \). Constraints for signal and messages can be found in [18].

\[
\begin{align*}
\sum_{t_k \in T} A_{i,k} &= 1 \\
X_{i,j,k} &\leq A_{i,k}, \quad X_{i,j,k} &\leq A_{j,k}, \quad X_{i,j,k} + 1 &\geq A_{i,k} + A_{j,k}
\end{align*}
\]

Placement constraints are divided into two subsets. The first concerns resource utilization and consists in meeting the maximum capacity utilization of all execution nodes and communication buses. The second relates to allocation, and includes: (i) fixed allocation, when a function must be allocated to a specific node (e.g. a function responsible for collecting data from sensors has to be placed on the node linked to the sensor); (ii) exclusive allocation enforcing the placement of each function on exactly one node and each signal on one bus at most; and (iii) bus allocation enforcing the mapping of a signal on the bus connecting the nodes on which its sender and receiver functions are executed. This last constraint needs to take into account the topology of the communication network (e.g. if \( c_1 \) and \( c_2 \) are not connected then communicating functions \( f_1 \) and \( f_2 \) cannot be placed on \( c_1 \) and \( c_2 \), respectively). Function \( f_i \) is placed on node \( c_j \) if and only if it is placed on task \( t_k \) allocated to node \( c_j \). The parameter \( T[N][j] \) represents the set of tasks slots for \( c_j \). \( A_{n,j} \) is a boolean variable which denotes whether \( f_i \) is placed on \( c_j \). \( X_{2,i,j,k} \) is another boolean variable, with value 1 if \( f_i \) and \( f_j \) are placed on \( c_k \) and 0 otherwise. The following constraints ensure the consistency of the definitions of \( X_{2,i,j,k} \).

\[
\begin{align*}
A_{n,j} &= \sum_{t_k \in T[N][j]} A_{i,k} \\
X_{2,i,j,k} &\leq A_{n,k}, \quad X_{2,i,j,k} &\leq X_{n,k}, \quad X_{2,i,j,k} + 1 &\geq A_{n,k} + A_{n,k}
\end{align*}
\]

Signal \( s_i \) is placed on bus \( b_j \) if and only if it is placed on the message slot \( m_k \) belonging to \( b_j \). \( MB[j] \) represents the set of messages slots belonging to \( b_j \). \( Ab_{i,j} \) is a boolean variable which denotes whether the signal \( s_i \) is transmitted over the bus \( b_j \).

\[
Ab_{i,j} = \sum_{m_k \in MB[j]} A_{i,k}
\]

\( G_i \) is a boolean variable equal to 1 if \( s_i \) is transmitted on a bus and 0 if \( s_i \) is a local signal; \( NB[j] \) is the set of nodes communicating through bus \( b_j \). The following constraints enforce the conditions for \( s_i \) to be on \( b_j \).

\[
\begin{align*}
Ab_{i,j} &\leq G_i, \\
0 &\leq \sum_{e_h \in NB[j]} An_{rec_i,k} + G_i + \sum_{e_h \in NB[j]} An_{send_i,k} - 3 \cdot Ab_{i,j} \leq 2
\end{align*}
\]

where \( snd_i \) and \( rec_i \) are, respectively, the source and destination function of the signal \( s_i \).

Execution order constraints are local total order and functional order constraints. The local total order constraints consist
in a total priority order for tasks/messages belonging to the same node/bus. However, within each node and bus we assign a different priority order to task \( t_i \) and message \( m_j \) slots, which is denoted respectively as \( \pi_i \) and \( \tau_j \). Functional order constraints represent partial orders of execution (such as \( t_i \) may not be executed before \( t_k \) when \( t_i \) depends on the output of \( t_k \)). Tasks dependencies are derived from the dependencies of the functions mapped in them. The (constant) parameter \( dp_{\tau,j} = 1 \) indicates that the execution of \( f_j \) depends on \( f_i \) in the functional graph. In this case, \( f_j \) cannot be placed on a task with priority higher than the task slot on which \( f_i \) is placed.

\[ \forall f_i, f_j \ s.t. \ dp_{\tau,i,j} = 1 : \sum_{k \in \Gamma} A_{j,k} \cdot \pi_k \leq \sum_{k \in \Omega} A_{i,k} \cdot \pi_k \]

**Response time constraints** are common to the placement, partitioning, and scheduling sub-problems. Each transaction \( \Gamma_i \in \zeta \) has a latency \( L_i \) equal to the worst-case response time (WCRT) \( R_{\Gamma_i} \) of its last function \( f_j \) and the latency of each transaction should not exceed its deadline \( D_i \leq D_{f_i} \). To compute the WCRT of functions, we adapt the response time analysis with jitter propagation (as applied to tasks) given in [28] and [21].

The WCRT \( R_{\Gamma_i} \) of a function \( f_i \) is computed by considering all the \( q \) function instances (distinct executions of \( f_i \) after activation) in the busy period, as follows:

\[ R_{\Gamma_i} = \max \{ W_{f_i}^{(q)} - (q - 1)P_{f_i} + J_{f_i} \} \quad (1) \]

Since \( f_i \) is executed by a task, its release jitter \( J_{f_i} \), is the task release jitter, that is, the largest among the latest release times for the functions in the same task, which is zero if the function \( f_i \) has no predecessor (or a predecessor within the same task), or the worst case response time of the signal it receives from a remote predecessor function \( s_i \) is the signal received by \( f_i \).

\[ J_{f_i} = \max_{s_i = s_{f_i}, \forall f_j \in \Phi \{ 0, R_{s_i} \}} \]

\[ W_{f_i}^{(q)} \] is the completion time of the \( q \)th instance of function \( f_i \). \n
\[ W_{f_i}^{(q)} = q \cdot (\omega_{f_i,c_k} + \sum_{f_j, \tau_j = \tau_f} \omega_{f_j,c_k} + \sum_{f_j \in hp(f_i)} \frac{W_{f_j}^{(q)} + J_{f_j}}{P_{f_j}}) = \omega_{f_i,c_k} \quad (3) \]

where \( hp(f_i) \) refers to the set of higher priority functions than \( f_i \) executing on the same node and within different tasks. The last term represents the preemption time from functions belonging to \( hp(f_i) \). The completion time is computed for \( q = 1, 2, \ldots \) until the busy period ends, that is, an instance completes at or before the activation of the next instance.

When a (higher priority) task contains functions with different periods, its interference is computed as the sum of the interferences of its functions, as shown in Figure 2, in which a task consists of two functions \( f_1 \), activated once every two task executions, and \( f_2 \) executed every time. The exact response time formula (3) is not suitable for an MILP formulation. It depends on the number of function activations \( q \) in the busy period, which is not known a-priori. Hence, we use a necessary-only (optimistic) conditions for feasibility that only considers the response time of the first instance \( (q = 1) \). The MILP formulation of (3) with \( q = 1 \) becomes (the integer variables \( I_{f_i} \) represent the number of times \( f_i \) may interfere with \( f_i \) in the worst case)

\[ W_{f_i} = \omega_{f_i,c_k} + \sum_{f_j, \tau_j = \tau_f} \omega_{f_j,c_k} + \sum_{f_j \in hp(f_i)} I_{f_j} \cdot \omega_{f_j,c_k} \quad (4) \]

The function response time is equal to its first instance (as computed by the above formula) in case of restricted deadline \( L_i \leq D_i \leq P_{f_i} \). When end-to-end deadlines may be larger than the periods or the interarrival times of activation events, the formulation using the constraint (4) may compute an optimal solution that is not feasible. This is why all the solutions obtained from the MILP solver must be verified afterwards using the exact response time formula.

The WCRT \( R_{s_i} \) of a signal \( s_i \) is computed only for signals representing remote communications (otherwise it is equal to zero), using a similar formula as the WCRT of functions, except for an additional term \( B_{s_i} \) that represents the blocking time due to the impossibility of preempting messages. The release jitter of a remote signal is the worst-case response time of its sender function.

Constraint (5) represents the computation of \( B_{s_i} \) of a signal \( s_i \) that is computed only for signals representing remote communications (otherwise it is equal to zero), using a similar formula as the WCRT of functions, except for an additional term \( B_{s_i} \) that represents the blocking time due to the impossibility of preempting messages. The release jitter of a remote signal is the worst-case response time of its sender function.

\[ \forall s_j \in \Phi : m_{s_j} = m_{s_i}, B_{s_i} = \omega_{s_j,b_k} + \sum_{s_k : m_{s_k} = m_{s_i}} \omega_{s_k,b_k} \quad (5) \]

**Optimization metrics** can be defined based on the system requirements. In this work we tried two formulations: (i) the minimization of the sum of all (or some) transactions latencies, which is a loose indication of the system performance, and (ii) the maximization of the minimum transactional slack time. A slack time for a given transaction is defined as the difference between the deadline and latency of the transaction. Maximizing the minimal transactional slack time (over all transactional slack times) means maximizing the minimum distance between the latency and deadline of the selected transaction \( \min \{ \text{Slack} = \min_{f_i \in \zeta} (D_{f_i} - L_{f_i}) \} \). This latter metric can be easily related to the concept of robustness (or extensibility) of the system against changes in the time parameters of some functions.

### 4. Two-step optimization for PPS

The (one-step) MILP formulation of the PPS problem can only be solved for small systems. To handle industry-size systems, the problem is divided in two smaller (sub)problems and solved in cascade. Two MILP formulations are then provided, one for each sub-problem. A GA counterpart for each sub-problem is also given.

#### 4.1 Overview

Our heuristic (algorithm in Figure 3) combines two classical optimization strategies: divide-and-conquer and iterative improvement. **Divide-and-conquer** consists in dividing the PPS problem in two sub-problems solved in cascade, in which placement is solved first (placemnt problem or \( PP \) ) and, then partitioning and scheduling (\( PS \)). **Iterative improvement** is used to move towards the optimum. Our algorithm considers an iterative improvement at two levels: inner and outer loops. The inner loop tries to find an optimal system
configuration by applying iteratively an optimization sequence until convergence (two successive solutions are the same). The inner loop consists of the two stages of placement optimization (PP), followed by partitioning and scheduling optimization (PS). In both sub-problems, we aim at optimizing the latency- or slack-based metric. Each iteration starts from a PP step with an initial PS configuration. The PP step provides a new placement of functions/signals to nodes/buses and so are the functions mapped onto them. Next, the PS step tries to find a new partitioning and scheduling solution that improves the solution found in the PP stage. Depending on the selection of the initial configuration, the PP+PS solution may be a local optimum. To move away from local minima, the outer loop selects set of random initial configurations for partitioning and scheduling as possible starting points.

![Figure 3. Overview of the Two-step optimization approach](image)

### 4.2 The MILP formulation for the PP and PS stages

This section outlines the MILP formulation for each sub-problem in the two-step approach.

The two-step formulation uses the same optimization metric for the one-step formulation (Section 3.2).

#### 4.2.1 MILP formulation of the Placement Problem

The objective of this step is optimizing the placement of either an initial configuration or the output of the previous iteration. The parameters are the partitioning of functions (signals) on tasks/messages and the priorities assigned to tasks and messages. This objective is the optimal placement of tasks/messages and the function/signals in them.

**Placement constraints**: as mentioned before, they concern allocation and resource utilization constraints. \(C[i]\) in (6) is the set of nodes on which \(t_i\) is allowed to execute (this includes the case of fixed allocation). \(At_{i,c}\) is a binary variable set to 1 if a \(t_i\) is placed on node \(c\). Each task must be placed on one node.

\[
\sum_{c \in C[i]} At_{i,c} = 1 \tag{6}
\]

The binary variable \(H_{i,j,c}\) based on \(At_{i,c} \cdot H_{i,j,c}\) denotes whether \(t_i\) and \(t_j\) are placed on the same node.

\[
0 \leq At_{i,c} + At_{j,c} - 2 \cdot H_{i,j,c} \leq 1 \tag{7}
\]

Messages placement is based on the placement of tasks. A message can be placed on one bus or to no bus (in case of local communication). The binary variable \(g_i\) indicates if \(m_i\) is placed on a bus \((g_i = 1)\) or is local \((g_i = 0)\). The binary variable \(Am_{i,j}\) is 1 if \(m_i\) is transmitted on bus \(b_j\), 0 otherwise.

\[
\sum_{b_j \in B} Am_{i,j} = g_i \tag{8}
\]

The utilization of buses is constrained in a similar way. The maximum utilization of nodes is enforced by constraint (11).

\[
\sum_{c} H_{i,j,c} \cdot H_{i,j,c} \leq M \tag{11}
\]

#### 4.2.2 MILP formulation for the Partitioning and Scheduling

At this stage, the placement of functions and signals is given and the following MILP formulation aims at improving (if possible) the partitioning and priority assignment. The MILP formulation assigns priorities to functions and signals within each node and bus, then tasks and messages are constructed based on priorities, i.e. a task (message) is the set of functions (signals) with the same priority order residing on the same node (bus).

**Partitioning and scheduling constraints**: most constraints are expressed in the same way for functions and signals. In the following, we address only functions and discuss the differences for signals. The binary variable \(\chi_{i,j}\) indicates whether \(f_i\) has higher priority than \(f_j\). If \(\chi_{i,j} = 0\) and \(\chi_{j,i} = 0\) then \(f_i\) and \(f_j\) have the

\[
\forall (t_i \in \text{sen}(g_i), t_j \in \text{rec}(g_j) : 1 - \sum_{c \in C} H_{i,j,c} = g_i \tag{9}
\]

Since all communicating tasks residing on different nodes must have a connecting bus, the placement of messages must also take into account the network topology. The parameter \(NB[j]\) is the set of nodes communicating through the bus \(b_j\). A message \(m_i\) is placed on \(b_j\) iff the execution nodes of its sender and receiver tasks communicate through \(b_j\).

\[
\forall (t_i \in \text{sen}(g_i), t_j \in \text{rec}(g_j)) : 0 \leq g_i + \sum_{c \in NB[j]} At_{i,c} + \sum_{c \in NB[j]} At_{j,c} - 3 \cdot Am_{i,j} \leq 2 \tag{10}
\]

\(H_{i,j,k}\) indicates if \(m_i\) and \(m_j\) are transmitted on the same bus \(b_k\). The maximum utilization of nodes is enforced by constraint (11). The utilization of buses is constrained in a similar way.

\[
\forall c \in C : \sum_{i} A_{i,j,c} \cdot \left(\omega_{i,c} / P_i\right) \leq \mu_j \tag{11}
\]

**Latency constraints**

The task WCRT is the sum of its completion time and jitter.

\[
R_i = W_i + J_i \tag{12}
\]

The Big M method is used in (15) and (16) to linearize the computation of \(I_{i,j,c}\), the maximum number of times that \(t_j\) preempts \(t_i\) as \(I_{i,j,c} = H_{i,j,c} \cdot \sigma_{i,j}\), where \(\sigma_{i,j} = \frac{W_i + J_i}{P_j}\). The binary parameter \(g_{p_{i,j}}\) denotes whether \(\pi_i\) is higher than \(\pi_j\).

\[
W_i = \sum_{c_p \in C} A_{t_i,p} \cdot \omega_{i,p} + \sum_{j \in T} \sum_{c_p \in C} I_{i,j,p} \cdot \omega_{j,p} \cdot g_{p_{i,j}} \tag{13}
\]

\[
0 \leq \sigma_{i,j} - \left(\frac{W_i + J_i}{P_j}\right) < 1 \tag{14}
\]

\[
\sigma_{i,j} - M \cdot (1 - H_{i,j,c}) \leq I_{i,j,c} \tag{15}
\]

\[
I_{i,j,c} \leq \sigma_{i,j} \cdot I_{i,j,c} \leq M \cdot H_{i,j,c} \tag{16}
\]

The jitter of a task is 0 if all its functions are initial and otherwise is equal to the largest WCRT of the received messages (17).

\[
\forall m_j \in \psi \ s.t. t_i \in \text{rec}(g_j) : J_i \geq R_{m_j} \tag{17}
\]

The constraints computing the WCRT of messages are similar to (12), (13), (14), (15), and (16) except for the addition of the blocking time \(B_{m_j}\) to the formula (13)

\[
\forall m_j \in \psi : B_{m_j} = \sum_{c \in C} H_{m_i,m_j,b} \cdot \omega_{m_j,b} \tag{18}
\]

Note that the WCRT of a message transmitted inside a node returns only its jitter as its completion time is zero. Thus the task jitter in this case is simply the WCRT of its predecessor.

The result of this MILP formulation consists in a placement of tasks and messages and the input for the next stage is the placement of functions and signals as partitioned in the tasks and messages.
same priority order, otherwise, either \( f_i \) has higher priority than \( f_j \) (\( \chi_{i,j} = 1 \)) or \( f_j \) has higher priority than \( f_i \) (\( \chi_{j,i} = 1 \)).

\[
\chi_{i,j} + \chi_{j,i} \leq 1 \quad (19)
\]

A set of constraints is used to enforce the symmetric, transitive and inversion properties of the priority order relation and to ensure that each function is partitioned on exactly one task (omitted for space constraints).

A binary variable \( SP_{f_i,f_j} \) denotes whether \( f_i \) and \( f_j \) have the same priority order i.e. they are on the same task.

\[
1 = SP_{f_i,f_j} + \chi_{j,i} + \chi_{i,j} \quad (20)
\]

Any pair of functions with non-harmonic rates must be assigned to different tasks (have different priorities).

The second part of run-time constraints applies only to signals: \( s_i \) and \( s_j \) cannot be partitioned on the same task if they belong to the same transaction (\( \Gamma_{[s_i]} = \Gamma_{[s_j]} \)).

\[
SP_{s_j,s_i} \neq 0 \quad \text{if } \Gamma_{[s_i]} = \Gamma_{[s_j]} \quad (22)
\]

In constraint (23) the parameter \( dp_{f_i,f_j} = 1 \) indicates that the execution of \( f_i \) depends on \( f_j \) in the functional graph. In this case, \( f_j \) cannot be placed on a task with priority higher than the task on which \( f_i \) is placed.

\[
\chi_{i,j} = 0 \quad \text{if } dp_{f_i,f_j} = 1 \quad (23)
\]

**Latency constraints**: constraints on latency are similar to the previous ones, except that, since tasks are variables in this stage, we compute the WCRT on functions while considering their partitioning. The WCRT of a function \( f_i \) is the WCRT of the task containing \( f_i \). Constraint (24) expresses the computation of functions WCRT. Signals WCRT are computed in the same way.

\[
R_{f_i} = W_{f_i} + J_{f_i} \quad (24)
\]

The first term of the functions completion time (25) represents the WCET of the task containing \( f_i \), as the sum of the WCET of its functions. The binary parameter \( SN_{f_i,f_j} \) indicates whether \( f_i \) and \( f_j \) are placed on the same node. The second term refers to the interferences from higher priority functions \( f_k \) that are on the same node as \( f_i, f_{i,k} \) and \( \sigma_{f_i,f_j} \) is a real variable representing the maximum number of times that \( f_k \) preempts \( f_i, f_{i,k} \) is computed as in (14) and \( I_{f_i,f_k} \) is linearized using the Big M method.

\[
W_{f_i} = \sum_{f_j \in F} SP_{f_i,f_j} \cdot \omega_{f_j} \cdot \omega_{f_j} \cdot SN_{f_i,f_j} + \sum_{f_k \in F} I_{f_i,f_k} \cdot \omega_{f_k} \cdot \cdot SN_{f_i,f_k} \quad (25)
\]

The response time of signals is computed similarly, adding a term corresponding to the blocking time \( B_{s_i} \) as in (26), where \( SB_{s_i,s_j} \) is a binary parameter indicating whether \( s_i \) and \( s_j \) are transmitted over the same bus. \( W_{s_i} \) is 0 if \( s_i \) is local.

\[
\forall s_j \in \Phi : B_{s_i} = \sum_{f_k \in F} SP_{s_i,s_j} \cdot B_{s_i,s_j} + \sum_{s_k \in \Phi, k \neq s} SP_{s_i,s_k} \cdot SB_{s_j,s_k} \quad (26)
\]

To compute functions jitter we need to define a real variable \( V_{f_i} \), which is 0 if \( f_i \) is the first function in its transaction, otherwise, it is the WCRT of the signal it receives.

\[
V_{f_i} = \begin{cases} 0 & \text{if } f_i \text{ is triggered by an external event} \\ f_i & \text{if } s_i \text{ is the signal received by } f_i \end{cases} \quad (27)
\]

\( J_{f_i} \) is computed as the largest \( V_{f_i} \) value among all functions with the same priority and belonging to the same node. As above, the Big M method is used to linearize \( V_{f_i} \cdot SP_{f_i,f_j} \).

\[
\forall f_j \in F \text{ s.t. } SN_{f_i,f_j} = 1 : J_{f_i} = V_{f_i} \cdot SP_{f_i,f_j} \quad (28)
\]

Similarly, we use an auxiliary real variable \( (X_{s_i}) \) to compute the jitter of signals, which depends on the placement and partitioning of its sender and receiver functions. In (29), the real variables \( Y_{f_i,f_j} \) and \( Z_{f_i,f_j} \) are equal respectively to \( J_{f_i} \cdot SP_{f_i,f_j} \) and \( R_{f_i} \cdot (1 - SP_{f_i,f_j}) \). These latter variables are also linearized using the Big M method. According to (29) and the above definition of \( Y_{f_i,f_j} \) and \( Z_{f_i,f_j} \), \( X_{s_i} \) may have two values depending on the variable \( SP_{f_i,f_j} \) and the parameter \( SN_{f_i,f_j} \). When the sender \( \text{(sent}_{s_i}) \) and receiver \( \text{(rec}_{s_i}) \) functions of \( s_i \) are on different nodes, or on the same node but different tasks, \( X_{s_i} = R_{f_i} \cdot X_{s_i} = J_{f_i} \) if the sender and receiver functions are on the same task.

\[
X_{s_i} = (Y_{f_i,f_j} + \sum_{f_j \in F} Z_{f_i,f_j} \cdot SN_{f_i,f_j} + R_{f_i} \cdot (1 - SN_{f_i,f_j})) \quad (29)
\]

Since a message may transmit multiple signals, the jitter of a signal should take into account the jitter of all signals transmitted on the same message. Constraint (30) defines \( J_{s_i} \) as the largest value of \( X_{s_i} \) among all signals belonging to the same message. If \( s_i \) is local, then its jitter is simply equal to \( X_{s_i} \).

\[
\forall s_j \in \Phi \text{ s.t. } SB_{s_i,s_j} = 1 : J_{s_i} = \max_{s_i} X_{s_i} \cdot SP_{s_i,s_j} \quad (30)
\]

This MILP formulation returns as output a set of functions (signal) partitions as well as a priority order for each partition (task or message) on each node (bus). As priority orders are given locally to a node (bus), a post-processing, which consists in defining a global priority order, is required before the next iteration. The global priority order is simply defined based on the order considered as input in the previous step of the same iteration.

### 4.3 The GA formulation for the two-step approach

Genetic Algorithm (GA) is an optimization technique patterned after natural selection in biological evolution (Algorithm 4.3.1). In a GA, the space of all possible solutions (feasible and not feasible) to the optimization problem is encoding using a string of bits, called chromosome. Each bit or group of bits in the sequence typically encodes one parameter of the solution (such as the placement of a function or the priority of a task). Several solutions are generated at each round (population), starting from an initial set and then obtaining new solutions by a composition function (or crossover) that applies to two chromosomes and produces a new one or by a mutation operator that changes the bit string of a chromosome to generate a new one. Each new generation (or offspring) is evaluated. Some bit strings correspond to non-feasible solutions, or dead individuals and are discarded. A set of the most promising ones is retained and used for computing the next generation.

In the context of our problem, we implement two GA algorithms for the PP and PS stages respectively. GA is used because of its better scalability, in comparison to MILP. Let us note that the quality of the solutions obtained from GA is difficult to evaluate and guarantee, since it is based on many factors, such as the choice of the encoding, crossover and mutation operators, the fitness function and additional mechanisms that improve the search and guarantee constraints.

#### 4.3.1 A GA Solution to the Placement Problem

The **Encoding** definition translates a solution configuration in a string of bits. In the placement problem, a specific solution, i.e. single chromosome, represents the allocation of tasks onto the processing units, and messages on buses. We used the value encoding, in which each gene (subset of bits) in a chromosome contains a specific value. In our case, a gene relates either to a task or a message. The value held by a gene represents an execution node or a communication bus. Although formally PP stage refers to the functions and signals placement here we are encoding tasks and messages. This is due to the knowledge about the functions and signals.
partitioning during the PP stage. Therefore the placement of functions and signals can be directly inferred from the tasks/messages allocation. There are two advantages of this encoding. First, the number of tasks/messages is never greater than the number of functions/signals. Hence the size of a chromosome will not be greater if functions/signals were encoded. This can significantly save memory which is an issue especially if large initial populations are considered. Secondly, in case of functions/signals encoding, additional mechanism to preserve the correctness of the chromosome in regards to the allocation constraints would be required. Namely, all the functions/signals of the same task/message have to be allocated on the same processor/bus.

Algorithm 1 General form of a GA Algorithm

1: // Define encoding, crossover and mutation operator, fitness
2: // Specify the size of an initial population - $P_{size}$
3: // Generate initial population
4: while termination condition is not met do
5:   Evaluate each solution from the population $P$
6:   Generate new population $P$ by applying the crossover and mutation operators
7: end while
8: return the best solution from $P$

Figure 4 shows an example of a chromosome that corresponds to the specific configuration for placement. Gene $t_1$ holds a value equal to 1, which is the index of the node on which $t_1$ and all its functions, i.e. $f_1$ and $f_2$ will run. The value of the gene $m_1$ is 0, indicating that message $m_1$ in this placement configuration is locally transmitted. The selection of the Crossover operator is very important for the quality of the GA solution. The operator combines information from two parent chromosomes to create a new child. The choice of the parent chromosomes can be done in many ways, but it is always highly dependent on a chromosome fitness rate. For our implementation we selected the OX3 crossover operator [11] with a tournament selector [19] (with size equal to 5). The OX3 creates two child chromosomes from two parents. It selects two random positions in a chromosome. The values between these points are copied from the first/second parent to the second/first child in the same absolute position. The remaining values are copied from the first/second parent to the first/second child. A simple result of the application of this operator on two chromosomes is shown on the Figure 5. The Mutation operator chooses a random point in a chromosome and changes the value of the gene at the selected point to a new random value. If the randomly selected gene corresponds to a task, the new value is chosen from the list of available execution nodes. If it relates to a message, the new value is chosen from the list of available buses.

The Fitness function defines how much the solution optimizes the performance criteria. Chromosomes are ranked according to this function and, the higher the rank, the higher the probability that the chromosome is selected as a parent for a crossover or the target of a mutation. Our fitness function corresponds to the optimization metrics, that is, in the case of the minimum transactional slack time, it computes $\min_{i \in \Gamma} (D_i - L_i)$ where $L_i$ is the latency of the transaction $\Gamma_i$ and $D_i$ its deadline.

The Initial population is generated randomly, i.e. for each task gene, a random number representing its execution node is assigned. However, the initial population does not contain solutions which violate the utilization constraints. Therefore if a generated chromosome leads to the violation of a utilization constraint, we call a correction procedure.

Correction mechanisms are used to avoid the generation of non-feasible solutions in the initial population or after the crossover and mutation. In the case of the violation of utilization constraint, the chromosome is modified by lowering the load of the node(s) with excessive utilization. The procedure randomly selects a task from one of these nodes and then moves it to a destination node, randomly selected among those that can accommodate the additional load. Tasks are moved until a feasible load distribution is found. Incorrect definitions of the communication are also fixed. If two communicating tasks are placed on different nodes, the gene in a chromosome that relates to the message exchanged between the tasks must have a number associated with one of the buses that connect the two nodes. Our correction mechanism checks all message values. Each time an incorrect bus is found, the procedure randomly generates a new bus identifier among those that are valid with respect to the tasks placement.

4.3.2 The GA Formulation for Partitioning and Scheduling

After the definition of the function and signal placement (implicitly by the placement of tasks and messages), the maximum number of new possible tasks and messages for each node and bus can be computed as the number of functions or signals allocated on the resource. Also, signals that result in local communications are not represented in chromosomes. For the PS stage, we only describe the encoding, the generation of the initial population, and the correction mechanism. The crossover mutation operators follow the same logic as in the placement stage. The fitness function is the same.

In the PS Encoding each gene represents a function or a signal exchanged among CPUs. The value of the gene is the index of the task or message executing the function or transmitting the signal. The index of a task or message also represents its priority, and its period is the gcd of the functions/signals mapped onto it. In the case of Figure 4, the system partitioning and scheduling is represented by the chromosome shown in Figure 6, where $f_1$ is executed by $t_1$ (with priority 1), together with function $f_2$.

Figure 6. Chromosome for the partitioning and scheduling configuration

The Initial population is randomly generated by assigning a task or message index to each function and signal.

The Correction function, called when a new chromosome is generated as part of the initial population or after the crossover and
mutation enforces the order of execution constraints. The range of values for a gene is constrained by the values assigned to other genes. If function \( f_1 \) precedes \( f_2 \), and the gene representing \( f_1 \) is assigned to a task with priority \( \pi_1 \), then \( f_2 \) should be partitioned on the same task or a task with priority lower than \( \pi_1 \).

5. Experiments

The objective of the following experiments is to assess (i) the quality of solutions obtained with the two-step technique (MILP and GA) against optimal solutions given by the one-step MILP, (ii) comparison of the two-step MILP versus the two-step GA when applied to an industry-size system and (iii) scalability and runtime evaluation of all the techniques.

5.1 Quality evaluation of two-step techniques against one-step MILP

This section presents a first set of small-size tests with the same initial configuration. Results show the convergence of the inner loop and the impact of the initial configuration on the quality of results. The second experiment applies the two-step and the one-step techniques to an automotive case-study. For this case three different initial configurations are tested.

5.1.1 Small-size tests

To evaluate the quality of solutions given by the two-step techniques, we choose six randomly generated systems with functional and signal WCETs and the WCTTs of signals are the same for all nodes and buses, and the network topology consists of two nodes and a single bus. The maximal capacity utilization is set to 1 for the bus and both nodes. The initial configuration for partitioning and scheduling is as follows: (i) each function executes in one task and each signal is transmitted by one message; (ii) priorities are assigned to tasks as follows: (1) if they belong to the same transaction, if the function \( f_i \) depends on the function \( f_j \) then \( t_i \) is higher priority than \( t_j \), (2) between transactions, we follow the deadline-monotonic (DM) approach [5]. Messages inherit the priority of the sending task. Figure 7 shows the comparison of the optimal solutions with the solutions provided by the two-step techniques (placement is represented by full lines, partitions are represented by dashed lines and the higher index of tasks represents the higher priority). For these tests the two-step techniques for both MILP and GA computed the same results. For tests 3, 4 and 5 the two-step techniques (MILP and GA) computes the optimal solution Figure 8(b). For tests 1, 2 and 6 they compute a local optimum Figure 8(a). In the three last cases, the selected initial configuration prevents finding the optimum. The two-step techniques return the solutions for which interferences between tasks are minimal. This results in splitting functions between nodes, which in next iterations prevents finding the solution in which all functions are on the same task and node. This situation does not occur for the test 3 where WCTTs are very large with respect to tasks interferences.

![Figure 7](image_url)

Figure 7. Tests for evaluating solutions quality

![Figure 8](image_url)

Figure 8. Comparison results between two-step and optimal solutions for tests 1-6

Table 1 shows partial results obtained at each step of the inner loop. For these tests, two-step MILP and GA computed the same results at each step of all iterations. The values in the cells represent the maximum of the minimum transactional slack time (\( O_2 \)) and the minimum sum of latencies (\( O_1 \)). At each step a better or equal solution is found. Convergence is obtained in the worst case at the third iteration.

5.1.2 Automotive case study

We consider an automotive system composed of the CCS and ABS sub-systems [4][20]. Figure 9 represents the functional graph of this automotive system. Each function (signal) has the same WCET (WCTT) for all nodes (buses). The network topology for this test is composed of four nodes and a single bus. The maximal capacity utilization is 1 for the bus and all nodes. For this system, we define three initial configurations for partitioning and scheduling as shown in Figure 10. Partitions are represented by dashed lines and
the index of the partition defines its priority (higher index means higher priority). The optimal solution given by the one-step MILP is shown in Figure 11 (right-most solution). The values for the minimal slack and latency are respectively 7.45 and 68.27. Table 2 provides the results obtained with the two-step MILP at each iteration of the inner loop for the three selected initial configurations. Cell values refer to the maximum of the minimum transaction slack time ($O_s$) and the minimum sum of latencies ($O_l$). The obtained configurations are shown in Figure 11 for the MILP and in Figure 12 for the GA. GA computed different solutions with the same metric value and number of iterations for the first and second configurations. For the third configuration, the same solution was computed by MILP and GA. The optimal solution for this system is the one obtained with the two-step starting with the third configuration. Let us remark that considering different initial configurations in the outer loop allows moving towards better solutions. Other existing optimization heuristics [31] [22] consider only one random initial configuration.

5.2 Two-step techniques evaluation on an industrial-size system

The objective of the following experiment is to study the behavior of the two-step MILP and GA in the case of a large, distributed industrial-size system as in [6], but where the partitioning is not fixed. Figure 14 shows the functional graph of the system. The system has 9 nodes communicating through a single bus. The maximum capacity utilization is 1 for the bus and nodes. Nodes are heterogeneous and have different computation speeds. The WCETs and nodes allocation constraints for this system are given in Table 3. The WCTT for all signals is 10. In the initial configuration each function is partitioned in one task and the task priority is inversely proportional to the index of the function in it (i.e. the highest priority task contains the function with the smallest index).

The solution obtained with the two-step MILP for the first iteration is shown in Figure 13. The minimal slack for the two-step MILP is 2396 for the PP step and 3260 after the PS step (for the first iteration). The two-step GA computes slightly worse results, i.e. a minimum slack of 2387 for the PP step and 3262 for the PS step. Latency (in the first iteration) is the same for both methods, 9981 for the PP, and 6251 for the PS step.

5.3 Runtime and scalability evaluation

In this section we show respectively (i) runtime evaluation of all the techniques, pointing out the limits of one-step MILP when the size of the system grows and (ii) runtime evaluation of the two-step techniques for large systems, showing their scalability.

5.3.1 Runtime evaluation

In the first experiment, we are interested in showing the limits of the one-step MILP. We generate systems of 5,10,15 and 20 functions, by combining the transactions of Figure 7. The network topology is composed of four nodes communicating by two buses, the first bus connects nodes 1, 2 and 3 and the second nodes 3 and 4. The maximal capacity utilization is 1 for all nodes and buses. We run two outer loops for two-step techniques. We compare the quality of the solutions for all configurations.
of solutions returned by one-step and two-step techniques, then we study the time required to compute the solution.

![Figure 13. Deployment solution for the large case](image)

![Figure 14. Functional graph of the large case](image)

functions. The runtime of the two-step techniques (for two outer loops) is very small compared to the one-step MILP runtime, and the one-step MILP runs out of memory (OOM) for systems with 20 functions, while the two-step MILP computes a solution in 1375 seconds. Let us note that in all the cases the one-step MILP returns without error, the obtained values of latency are the same for all algorithms.

### 5.3.2 Scalability evaluation of two-step techniques

To study the scalability of the two-step techniques, we generated systems of different sizes by combining the transactions shown in Figure 14. We consider only the inner loop iteration for two-step techniques (one initial configuration). The obtained results are shown in Table 4, where n/a means that there is no solution from the MILP solver, (*) that the solver runs out of memory, (##) that the a solution is obtained after the PP stage, but the solver runs out of memory in the PS stage. The table shows that the two-step MILP (2-MILP) is more scalable (up to 80 functions) than the one-step MILP (1-MILP, up to 40 functions), but the 2-GA is the most scalable approach.

![Figure 15. Performance of the two-step optimization technique vs the one-step approach](image)

The results of the runtime comparison are shown in Figure 15: the runtime of all techniques increases according to the number of

### Table 3. Vectors of WCETs

| \( f_1 \) | 48 26 35 50 36 40 45 27 28 | \( f_{26} \) | 48 36 |
| \( f_2 \) | 30 32 39 34 26 43 50 48 49 | \( f_{27} \) | 30 50 |
| \( f_3 \) | 49 51 50 31 39 44 |
| \( f_4 \) | 26 53 29 32 30 51 28 49 50 |

![Table 4. Results on systems with different sizes](image)

<table>
<thead>
<tr>
<th>Nb functions</th>
<th>1-MILP</th>
<th>2-MILP</th>
<th>2-GA</th>
<th>1-MILP-2-MILP</th>
<th>2-GA</th>
</tr>
</thead>
<tbody>
<tr>
<td>32</td>
<td>3500</td>
<td>3430</td>
<td>3490</td>
<td>10278## 6557</td>
<td>55312</td>
</tr>
<tr>
<td>40</td>
<td>3385*</td>
<td>3462</td>
<td>3490</td>
<td>21533* 815</td>
<td>83052</td>
</tr>
<tr>
<td>50</td>
<td>n/a</td>
<td>3269</td>
<td>3262</td>
<td>n/a 1744 1345</td>
<td></td>
</tr>
<tr>
<td>60</td>
<td>n/a</td>
<td>2947##</td>
<td>3131</td>
<td>n/a 752## 1862</td>
<td></td>
</tr>
<tr>
<td>80</td>
<td>n/a</td>
<td>926##</td>
<td>3180</td>
<td>n/a 1260## 266795</td>
<td></td>
</tr>
<tr>
<td>100</td>
<td>n/a</td>
<td>n/a</td>
<td>3017</td>
<td>n/a n/a 351611</td>
<td></td>
</tr>
<tr>
<td>200</td>
<td>n/a</td>
<td>n/a</td>
<td>2030</td>
<td>n/a n/a 160523</td>
<td></td>
</tr>
</tbody>
</table>
Please note that other approaches solving task placement and scheduling [6] [30] [25] do not deal with more than 50 tasks and 9 nodes.

6. Conclusions

We presented two approaches to optimize the deployment of (hard) real-time distributed systems w.r.t end-to-end latency metrics. We consider an event-triggered activation model and define the placement, partitioning and scheduling of functions and signals. We provided an MILP integral formulation for the problem, which can compute the optimal solution for small size systems and a dual formulation, based on MILP and Genetic algorithm (GA) for a partitioned, two-stage iterative approach. We evaluated the performance of the two-step approach by comparing the results against the optimum for some small systems and then applied to larger case studies, including an automotive system. Future work includes an improvement of the MILP formulation for application to larger systems, an extension to more complex functional graphs, in which transactions are non-linear, and finally, the consideration of other optimization metrics.

References